micro sec 6

loop

DJNZ Reg, lablel

لما بعمل (decrement) لل reg ده بـ check كل

شوية ، طول ما هو مش بصفر بعمل (Jump) لل (label)

ex Reg = value

بعمل (decrement) لل (reg)
طول ما هو ليس بصفر بعمل
(loop)

```
Reg = value
---
----
----
---
DJNZ Reg, loop
```

— أكبر قيمة لل (reg) هنا FF (255)

لو لو عندي (loop) تحتاج قيمة أكبر من (255) هعمل
أكتر من . (loop) جوه بعض (nested loops)

— لو عندي مثلاً 700 . هعمل (2 loops)
واحده 10 والتانية 70 ، والناتج يكون حاصل ضربهم .

Jump

→ Conditional Sump

لذ بعمل (operation) ثم أعمل (check flag) وعلى
أساسه بعرف أخد أي (action) في البرنامج.

ex: JZ lablel

← بعمل (check) على (zero flag) ومنها أعمل
Jump لـ label.

if equal to zero Jump to label

else if not equal to zero ...

في المنطقة (*) هعمل (unconditional Jump)
لـ (label 2)

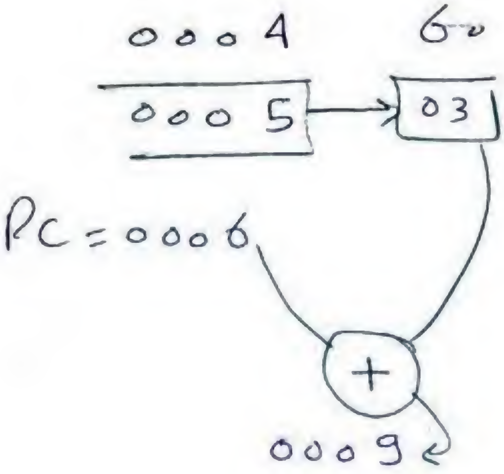
```
=====
JZ label
=====
* ====
label1 ====
=====

label 2
```

SJMP label →

انا بقدر أعمل (Jump) فوق أو تحت بحد أقصى (128 bit)

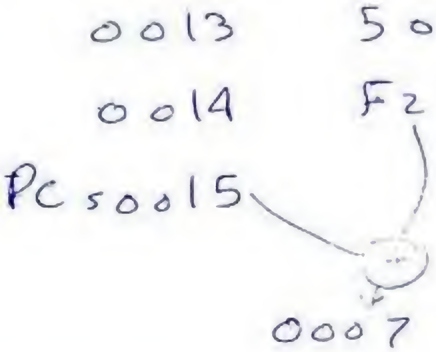
LJMP label (long Jump)

بعمل (Jump) لأي مكان في البرنامج.
لأنه (3byte) واحدة للـ (opcode) واتنين للـ (address)



look at page 102

at JZ Next



Call

بكتب شوية أكواد وفي النص عايز اعمل (subroutine)

فبعمله Call

لما بعمل (Jump) بكمل الكود بتاعي
لكن مش بترجع للبداية .

انما في ال (Call) بستخدم
امر (return) ترجع بيه
للبداية.

Call sub 1

sub 1:

RET return

- من مهام العمليات لأمر (Call) انه يعمل Push لـ (PC) ويعمل تغيير لـ (PC) لـ (sub1) عشان يشاور ليه.

PUSH PC
change Pc to sub1

sub1: _____

RET ⇒ PoP PC

- قبل الجزء بتاع (sub1) حضع (HLT) معناها نهاية البرنامج.

Long Call (LCall)

يعمل (Call) في أي مساحة في البرنامج

Absolute Call (ACall)

- يعمل (Call) في حدود (2K-byte) فقط.

- مثال في صفحة 117.

← لو فى ال (main) وال (subroutine) بتستعمل نفس ال (registers) ال (Call) مش دوره انه يحمى ال (register) لانه بيتعامل مع ال (Push) وال (return) فقط.

← فى الحاله دى قبل بدايه ال (subroutine) هعمل (Push) للقيم بتاعه ال (register) اللى هستخدمها وفى الآخر هعمل (PoP) ليهم بس بالعكس.

← لو عملت (Push) وفى الآخر مفيش (PoP) مش هيطلع قيم ال (Pc) العطلانة.

← لازم كل (Push) فى البدايه يقابلها نفس ال (PoP)

ال (inst.) يتم تقريبه بال (Crystal oscillator) و (Machine cycle)

↳ (cycle) يتم تنفيذ ال inst. فيها.

لما ال (MicroControllers) بداخله مجموعه من ال (operations) كل (operation) بتحتاج كام (Cycle)

→ # of clocks.

← ال (machine cycle) لل (atmel) عبارة عن (12 Clocks).

→ look at Page 122.

لم يستخدم الموضوع ده في أي عمل (delay)

لم ▪ المثال في صفحة 123.

LCD

مثال للإنك
← نستخدم (Hw) جداها (μc).

عبارة عن byte
← نبعتله (data) تتعرض عليه و (commands) يعرض بيها ال (data).

← ليها 14 Pin

14 - Pin
- → 8 - data
- → 3 : V_{cc} supply
  - V_{BB} Contrast
  - V_{DP} ground
- → 3 : E → enable.
  - R/W
  - Rs → 0 Command
  - Rs → 1 data

ال (LCD) اشتغال إزاي؟

عشان تتعامل مع ال (LCD)

- له نبعت ليه (data) و (Commands) اشتغل بيه ال (data)
- نبعت ليه (Commands) وبيبقى وبيخلي (Protocol) تتعلم بيه ال (Commands) دي.

R/W → 0 write
→ 1 read.

- لو بعت حاجة من ال (MC) لل (LCD) وبقوله اقرأيها المفروض ده بيعمل بال E (enable).

كل الحروف اللي انتا بتبعته عشان يتعرض بيتخزن في (Memory)

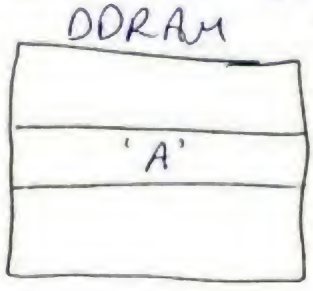
LCD Memory

→ DDRAM ,

- كل (character) بتتخزن لازم يكون ليه مقابل في ال (memory)

~~يعني لو عايز تعرض حاجة في الشاشة~~

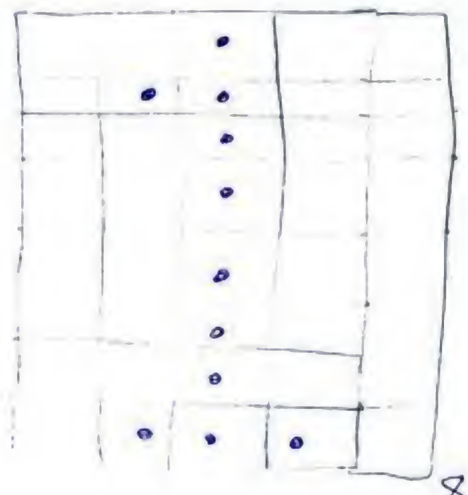
ex show 'A'
row 1 Col 3



الكنترولر هو اللي بيتابع حرف ال A.
بس هو لازم يعرف حرف A تترسم ازاي وده وظيفة (CGROM)

CGROM ← تعرف كل (ASCII code) بيترسم ازاي بيكون متخزن

فيها شكل رسمة الحروف.



← شكل حرف "1".

8 * 5

CGRAM

← هتوله اخزن فيها قيمة بس هحطها أنا اللي هعمل ال (Array) بنفسي.

← لو عايز أعمل (create) لحرف جديد مش موجود ← (special character).

ال task

← (Calculator) بتعمل الأربع عمليات

جمع وطرح وضرب وقسمة. على ال (LCD)